REDSTONE software

Position Piece

Black-box vs. White-box Testing:
Choosing the Right Approach to Deliver Quality Applications

Overview

Within the automated testing world there are two predominate testing methodologies: black-box and white-box. This paper seeks to explore the pros and cons of both approaches and to identify when each approach should be used to ensure quality applications are delivered to market.

In the end, this paper concludes that while black-box testing has had its drawbacks in the past, innovative approaches to black-box testing make it the likely choice to deal with the ever increasing complexity of applications and delivers lower Total Cost of Ownership (TCO) and a better Return on Investment (ROI) to organizations.

Definitions

Black-box: This testing methodology looks at what are the available inputs for an application and what the expected outputs are that should result from each input. It is not concerned with the inner workings of the application, the process that the application undertakes to achieve a particular output or any other internal aspect of the application that may be involved in the transformation of an input into an output. Most black-box testing tools employ either coordinate based interaction with the applications graphical user interface (GUI) or image recognition. An example of a black-box system would be a search engine. You enter text that you want to search for in the search bar, press "Search" and results are returned to you. In such a case, you do not know or see the specific process that is being employed to obtain your search results, you simply see that you provide an input – a search term – and you receive an output – your search results.

White-box: This testing methodology looks under the covers and into the subsystem of an application. Whereas black-box testing concerns itself exclusively with the inputs and outputs of an application, white-box testing enables you to see what is happening inside the application. White-box testing provides a degree of sophistication that is not available with black-box testing as the tester is able to refer to and interact with the objects that comprise an application rather than only having access to the user interface. An example of a white-box system would be in-circuit testing where someone is looking at the interconnections between each component and verifying that each internal connection is working properly. Another example from a different field might be an auto-mechanic who looks at the inner-workings of a car to ensure that all of the individual parts are working correctly to ensure the car drives properly.

The main difference between black-box and white-box testing is the areas on which they choose to focus. In simplest terms, black-box testing is focused on results. If an action is taken and it produces the desired result then the process that was actually used to achieve that outcome is irrelevant. White-box testing, on the other hand, is concerned with the details. It focuses on the internal workings of a system and only when all avenues have been tested and the sum of an application's parts can be shown to be contributing to the whole is testing complete.

The Pros and Cons

Black-box: There are many advantages to black-box testing. Here are a few of the most commonly cited:

1. **Ease of use**. Because testers do not have to concern themselves with the inner workings of an application, it is easier to create test cases by simply working through the application, as would an end user.

2. **Quicker test case development**. Because testers only concern themselves with the GUI, they do not need to spend time identifying all of the internal paths that may be involved in a specific process, they need only concern themselves with the various paths through the GUI that a user may take.

3. **Simplicity**. Where large, highly complex applications or systems exist black-box testing offers a means of simplifying the testing process by focusing on valid and invalid inputs and ensuring the correct outputs are received.

But, for all of the benefits of black-box testing, many attempts to create black-box test systems resulted in several drawbacks that caused people to question the viability of the black-box approach. Some of the most commonly cited issues were:

1. **Script maintenance**. While an image-based approach to testing is useful, if the user interface is constantly changing the input may also be changing. This makes script maintenance very difficult because black-box tools are reliant on the method of input being known.

2. **Fragility**. Interacting with the GUI can also make test scripts fragile. This is because the GUI may not be rendered consistently from time-to-time on different platforms or machines. Unless the tool is capable of dealing with differences in GUI rendering, it is likely that test scripts will fail to execute properly on a consistent basis.

3. **Lack of introspection**. Ironically, one of the greatest criticism of black-box testing is that it isn't more like white-box testing; that it doesn't know how to look inside an application and therefore can never fully test an application or system. The reasons cited for needing this capability are often to overcome the first two issues mentioned. The reality is quite different.

White-box: Like black-box testing, there are distinct advantages to white-box testing. Here are a few of the most commonly cited:

1. **Introspection**. Introspection, or the ability to look inside the application, means that testers can identify objects programmatically. This is helpful when the GUI is changing frequently or the GUI is yet unknown as it allows testing to proceed. It also can, in some situations, decrease the fragility of test scripts provided the name of an object does not change.

2. **Stability**. In reality, a by-product of introspection, white-box testing can deliver greater stability and reusability of test cases if the objects that comprise an application never change.

3. **Thoroughness**. In situations where it is essential to know that every path has been thoroughly tested, that every possible internal interaction has been examined, white-box testing is the only viable method. As such, white-box testing offers testers the ability to be more thorough in terms of how much of an application they can test.

Despite these benefits, white-box testing has its drawbacks. Some of the most commonly cited issues are:

1. **Complexity**. Being able to see every constituent part of an application means that a tester must have detailed programmatic knowledge of the application in order to work with it properly. This high-degree of complexity requires a much more highly skilled individual to develop test case.

2. **Fragility**. While introspection is supposed to overcome the issue of application changes breaking test scripts the reality is that often the names of objects change during product development or new paths through the application are added. The fact that white-box testing requires test scripts to be tightly tied to the underlying code of an application means that changes to the code will often cause white-box test scripts to break. This, then, introduces a high degree of script maintenance into the testing process.

3. **Integration**. For white-box testing to achieve the degree of introspection required it must be tightly integrated with the application being tested. This creates a few problems. To be tightly integrated with the code you must install the white-box tool on the system on which the application is running. This is okay, but where one wishes to eliminate the possibility that the testing tool is what is causing either a performance or operational problem, this becomes impossible to resolve. Another issue that arises is that of platform support. Due to the highly integrated nature of white-box testing tools many do not provide support for more than one platform, usually Windows®. Where companies have applications that run on other platforms, they either need to use a different tool or resort to manual testing.

Necessary vs. Nice to have

When approaching automated testing it is important to understand what is necessary vs. what is nice to have. Both testing methodologies have their merit. To determine what approach should likely be used there are a few questions that every company should ask:

1. Who will use the application?
2. What parts of the application must be tested prior to release and why?
3. What language will my application be written in?
4. When are significant changes to the UI likely to be made and will the underlying code be affected?
5. Where is the application likely to be deployed?
6. How will the application be used?
7. Which platforms does the application need to support?

Looking at each of these questions, some of the answers that you might come up with are as follows:

1. End users
2. Any part of the application that will be exposed to the end user
3. Java using an Eclipse framework
4. Annually because our customers are always looking for a new, fresh experience
5. On the web
6. Customers will login via the web and will enter information through a series of screens
7. Web browsers like FireFox, IE, Opera and Safari

Based on the answers given one can make an educated decision around the type of tool that should be used. In the case of the application described above, a black-box tool would likely be more effective as it is user-centric in its approach, focuses on testing the interface rather than the underlying code and a black-box tool is more likely to support the multiple platforms required.

Reviewing the questions suggested above does expose a more fundamental question about the nature of testing itself. Why does anyone test? The question is not as silly as it sounds and it reveals a lot about the long-term viability of the two approaches being explored herein.

All companies test their applications prior to release because their customers are intolerant of bugs. So, it is out of a need to satisfy customers that testing is undertaken in the first place. Given this, logic follows that to test in a manner that reflects how a customer will use an application should be a prerequisite. If this is true, then, only when black-box testing has been applied to an application can testing truly be said to have been completed.

This is an interesting argument and one that is unlikely to sit well with white-box tool vendors. However, let's review what we know. Customers use GUIs, not code. They enter information or interact with an application (the input) in some fashion and wait to get something back (the output). If the process works and they get an acceptable result, they are happy. If not, they experience a problem. This is identical to the approach employed by black-box testing

tools. As such, it would seem logical that a black-box tool should be used. Now, that is not to say that white-box tools to do not have their place, it is simply to say that they do not provide sufficient test coverage on their own for an organization to say that an application has been fully tested.

White-box tool vendors would object strongly to the previous statement. They would cite their introspective capabilities as being superior to a black-box testing approach. However, what a white-box vendor cannot guarantee is that what occurs at the code-level will be properly displayed on the UI. After all, the property of an object might be set to "visible" but due to a fault in another part of the application or even due to a problem outside of the application, the object might not appear to the user. A white-box tool would record the state change as having passed because the "visible" property was successfully updated. The fact that the user cannot actually see what they are supposed to is largely overlooked. Even where white-box vendors have introduced elements of image-recognition to overcome the "visible" problem highlighted above, white-box tools still tend to suffer from an inability to deal with periodic differences in image rendering.

Although issues with white-box testing have been identified above it should not be treated as a foregone conclusion that black-box testing provide comprehensive application testing coverage. Quite the contrary, what is hopefully evident at this stage is that only when black-box and white-box testing methodologies are combined is comprehensive test coverage achieved. The reason for this is that both methodologies address different aspects of application testing itself. For black-box testing, the focus is on the user experience whereas white-box testing focuses on the internal and making sure that the application works as efficiently as possible (or at least as designed). Therefore, these two methodologies can be seen as complimentary and for organizations that have the budget and time available to take advantage of both, they should certainly do so.

Black-box Testing: A Necessary Step

So far this position piece has highlight the pros and cons of both black-box and white box approaches to test automation and discussed when each approach might be most appropriate. In the last section, the idea of time and money was introduced recognizing that organizations have limited quantities of both. With those constraints in mind it is prudent to revisit the discussion of which approach provides the best test coverage for the lowest TCO and greatest ROI.

Looking at the merits of both black-box and white-box testing what seems to stand out is that black-box testing is focused on the end user; that undertaking black-box testing is the best way of ensuring that those parts of the applications that will be exposed to the user work correctly. Combined with the ease of use, quicker test case development and simplicity, black-box testing represents a lower initial cost than white-box testing and delivers ROI in a shorter period of time.

Recognizing that to be the case, in the face of budgetary and time constraints, black-box testing must be considered a necessary steps in quality assurance process while white-box test represents a nice to have.

Selecting a Black-box Testing Tool

When selecting a black-box testing tool there are several things to look for:

1. What platforms does it support?
2. How resilient is it to differences in the way images are rendered?
3. What facilities exist to aid in image recapture?
4. How does it identify images (by coordinates or image recognition)?

By asking these questions and many more you will be able to identify the black-box tool that is right for you. We would recommend that you try Eggplant as it delivers all of the benefits of black-box testing while overcoming nearly all of the cited limitations.

Whichever tool you choose, hopefully you recognize the importance of black-box testing in ensuring you are delivering applications that have been tested as they will be used and meet customer needs.

Further Information

For more information on how Redstone can help your company visit:

www.redstonesoftware.com
or you can email at sales@redstonesoftware.com

US Tel: +1 952 873 6809
US Toll-Free: +1 800 891 3486
UK Tel: +44 1489 555500

REDSTONE software